Meta-analysis of the effect of consistency on success in early learning of programming

Saeed Dehnadi, Richard Bornat, Ray Adams

S.Dehnadi@mdx.ac.uk, R.Bornat@mdx.ac.uk, R.Adams@mdx.ac.uk
School of Engineering and Information Sciences, Middlesex University

Abstract. A test was designed that apparently examined a student's knowledge of assignment and sequence before a first course in programming but in fact was designed to capture their reasoning strategies. An experiment found two distinct populations of students: one could build and consistently apply a mental model of program execution; the other appeared either unable to build a model or to apply one consistently. The first group performed very much better in their end-of-course examination than the second in terms of success or failure. The test does not very accurately predict levels of performance, but by combining the result of six replications of the experiment, five in UK and one in Australia. we show that consistency does have a strong effect on success in early learning to program but background programming experience, on the other hand, has little or no effect.

1 Introduction

Programming is hard to learn. The search for predictors of programming ability has produced no significant results. The problem is international and longstanding.

It is a commonplace that some students find programming extremely easy to learn whilst others find it almost impossible. Dehnadi observed that some novices confronted by simple programming exercises gave rational but incorrect answers. Their answers were mechanically plausible: for example, assigning (in Java) the value of a variable from left to right rather than right to left, or moving a value in an assignment rather than copying.

This suggested to us that some novices may have been equipped with some abilities before the course started. In an experiment [7] Dehnadi administered a test made up of questions about assignment and sequence programs. The test was administered before the first week of an introductory programming course, without giving any explanation of what the questions were about. Almost all participants gave a full response. About half used a rational model which they applied consistently to answer most or all of the questions. The other half did not seem to use a recognisable model or appeared to use several models. The consistent subgroup had an 85% pass rate in the course examination, and the rest a 36% pass rate.

There were some deficiencies in Dehnadi's experiment, and most of the psychology of programming community were sceptical of his result, particularly when two experiments [5,36] appeared to refute it, and a review of three others [4] concluded that the test does not predict very much of the variance in levels of performance in the course examination. Those 'refutations', however, showed only that his test was not psychometric, in that it is ineffective in a population of programmers, and it is important to distinguish between evidence for the presence of an effect and measurements of its strength. This paper provides evidence for the claim that consistency affects performance, by a meta-analysis of six replications of an improved version of Dehnadi's experiment. The evidence shows that consistency is not simply the result of background programming experience, and that by contrast such experience has little or no effect on success.

2 Previous Work

The search for predictors of success in learning to program has turned up very little. Cross [6], Mayer and Stalnaker [15] and Wolfe [35] tried to use occupational aptitude tests to predict

successful candidates for software industry's employment. McCoy and Burton [18] said good mathematical ability was a success factor in beginners' programming. Wilson and Shrock [33] found three predictive factors: comfort level, mathematical skill, and attribution to luck for success or failure. Beise et al. [3] found that neither sex nor age is a good predictor of success in the first programming class. Rountree et al. [23] reveal that the students most likely to succeed are those who are expecting to get an 'A' grade and are willing to say so. Lister et al. [14,11,22] opined, in a multi-national project, that incapability of students in entry-level programming is due to lack of general problem-solving ability. Simon et al. [26,31,27] followed up the project and came to similar conclusions.

Johnson-Laird's notion of 'mental model' [32] lies behind this study, and much other research on programming learning. Kessler and Anderson [13] and Mayer [17] all stressed the significance of mental models. Du Boulay [9] catalogued the difficulties that novices experienced. Fix et al. [12] differentiated mental models of novices and experts. Perkins and Simmons [19] described novice learners' according to their problem-solving strategies as "stoppers", and "movers". Mayer [16] described existing knowledge as a "cognitive framework" and how new information is connected to existing knowledge. Dyck and Mayer [10] emphasised the necessity for a clear under-standing of the underlying virtual machine in novice's learning process. Putnam et al. [21] studied the impact of novices' misconceptions of the capabilities of computers. Someren and Maarten [29] found that mechanical understanding of the way the language implementations is the key to success.

This study is based on the notion of mental models of rational misconceptions. Spohrer and Soloway [30] found that just a few types of bug cover almost all those that occur in novices' programs, and in [28] they studied novice's background knowledge and their misconceptions. Schneiderman [25] blamed the different uses of variables. Samurcay [24] looked at different ways variables are assigned values through assignment statements and describes how how internal variables like initialisation and updating is harder for novice programmers. Du Boulay [9] identified misconceptions about variables based upon the analogies used in class, misconception of linking variables by assigning them to each other, misunderstanding of temporal scope of variables, forgetting about initialisations. Perkins et al. [20] identified misconceptions about the names of variables.

3 Dehnadi's initial experiment

Dehnadi's experience of teaching led him to believe that novices bring patterns of reasoning to the study of programming: some of them appear to use rational mechanisms, distinct from those taught in the course, to explain program behaviour. He designed a test [7] to see what would happen when students were confronted with programming problems before they had been given any explanation of the mechanisms actually involved in program execution. His questions each gave a program fragment in Java, declaring two or three variables and executing one, two or

1. Read the following statements and tick the box next to the correct answer in the next column.	**The new values of a and b are:**	**Use this column for your rough notes please**

`int a = 10;`
`int b = 20;`
 	
`a = b;`
	☐ `a = 30` `b = 0`
	☐ `a = 30` `b = 20`
	☐ `a = 20` `b = 0`
	☐ `a = 20` `b = 20`
	☐ `a = 10` `b = 10`
	☐ `a = 0` `b = 10`
	Any other values for a and b:
	a = **b =**	

Fig. 1. The first question in the test, a single assignment

three variable-to-variable assignment instructions, as illustrated in figure 1. The student was asked to predict the effect of the program on its variables and to choose their answer/s from a multiple-choice list of the alternative answers. There was no explanation of the meaning of the questions or the equality "=" sign that Java uses to indicate assignment. Except for the word "int" and the semicolons in the first column, the formulae employed would have looked like school algebra, but when the question asked about the "new values" of variables it hinted that the program produces a change.

Dehnadi had a prior notion of the ways that a novice might understand the programs, and prepared a list of mental models accordingly. The eight mental models of assignment that he expected subjects to use were M1-M8 and M11 from table 2. Questions with more than one assignment required a model of composition of assignment as well as a model of single assignments. Dehnadi recognised the three models of sequential composition shown in table 3.

The test was administered to 30 students on a further-education programming course at Barnet College and 31 students in the first-year programming course at Middlesex University. No information was recorded about earlier education, programming experience, age or sex. It was believed that few had any previous contact with programming and that all had enough school mathematics to make the equality sign familiar.

Despite the lack of explanation of the questions, most students gave a more or less full response. About half gave answers which corresponded to a single mental model of assignment in most or all questions; the other half gave answers which corresponded to different models in different questions, or didn't use recognisable models.

Table 1. T1 and second quiz binary result

T1	**Pass**	**Fail**	**Total**
C	22	4	26
I	8	15	23
B	4	6	10
Total	34	25	59
$\chi^2 = 13.944,\ df = 2,\ p < 0.001$ **highly significant**			

When the result of the test was correlated with the result of in-course examinations it was found that, especially in the later examination which examined technical skill more thoroughly, the consistent (C) group had an 85% pass rate, but the others (I and B together) only 36%. This was a large effect, compared to earlier research on predictors, and a χ^2 test showed that it was significant. The result was over-hyped by Dehnadi and Bornat [8], and perhaps as a result attracted a good deal of hostile attention.

3.1 Two apparent refutations

An experiment at the University of Århus (Denmark) by Caspersen et al. [5] used Dehnadi's test but found no effect of consistency on success. Wray [36] at the Royal School of Signals (UK) used Dehnadi's test but found no effect of consistency on success and, by contrast, found a strong effect of Baron-Cohen's SQ and EQ measures [2,1].

In the Århus experiment 124 subjects (87%) were assessed as C, 18 (13%) as notC, and the average failure rate in the course was 4%. The high proportion of consistent subjects and the extremely low failure rate makes this experiment different from all but one of the experiments included in our meta-analysis below. There is some reason to believe, as discussed below, that lenient examinations reduce the effect of consistency on success; a 96% pass rate can be seen as lenient for the population being examined. The statistical mechanism used in this experiment

was unable to correlate the test with success/failure statistics, and their decision to look for correlations with graded data would in any case have weakened the effect of consistency. Despite the difficulties of dealing with such a population, our meta-analysis includes an experiment at the University of York which on the face of it has an intake similar to that in the Århus experiment.

Wray's experiment refutes the regrettable claim in [8] that Dehnadi's test divides novices into programming sheep and non-programming goats. The test should not be considered psychometric since a normal programming course attempts to train novices to pass it. Wray used Dehnadi's test five months *after* the course ended. It could not be expected to deliver a meaningful result in those circumstances.

4 Improved experimental protocol

In response to criticism of the initial experiment some improvements were made to the test. Most importantly, the judgement of consistency was made explicit and repeatable. The number of models of assignment recognised was expanded to the eleven shown in table 2. Answers still correspond to a single tick in most cases, but the new model M10 allows a student to tick each answer which makes `a` and `b` equal. The models of composition were made explicit as shown in table 3. Answer sheets were produced, illustrated in figure 4, which identified the mental models apparently used in particular answers or patterns of answers; in particular this exposed the ambiguous nature of some answers caused by models of composition, and requiring multiple ticks.

Questions about background were added to the questionnaire. We asked about age and sex, about previous programming experience (yes/no and if yes, what languages used) and previous programming course attendance (yes/no). A mark sheet was produced, shown in figure 3, which exposed the assessment of consistency. A marking protocol was developed to resolve the ambiguities introduced by use of model S3: essentially, we mark ambiguous responses across each row and look for the column which maximises the number of marks. We note that this makes consistency easier to achieve and, if anything, may dilute its effect on success.

These improved materials were used in all of the experiments analysed below.

5 New experiments

In the hope of dispelling scepticism that consistency has a noticeable effect on success in learning to program, the improved experiment was repeated several times by collaborating experimenters: at the University of Newcastle (Australia); twice at Middlesex University (UK); at the University of Sheffield (UK); at the University of York (UK); at the University of Westminster (UK); at

Table 2. Anticipated mental models of `a=b`

Model	Description	Effect
M1	right to left move	a←b ; b←0
M2	right to left copy	a←b
M3	left to right move	a→b ; 0→a
M4	left to right copy	a→b
M5	right to left move and add	a←a+b ; b←0
M6	right to left copy and add	a←a+b
M7	left to right move and add	a+b→b ; 0→a
M8	left to right copy and add	a+b→b
M9	no change	
M10	equality	a=b
M11	swap	a⇆b

Table 3. Anticipated mental models of `a=b; b=a`

Model	Description
S1	`a=b; b=a` Conventional sequential execution
S2	`a=b` || `b=a` Independent assignments, independently reported
S3	`a,b=b,a` Simultaneous multiple assignment, ignoring effect upon source

Question	Answers/s		Model/s
	$a = 10$	$b = 0$	M1+S1
	$a = 20$	$b = 10$	(M1+S3)/(M2+S3)/(M3+S3)/
5.			(M4+S3)
	$a = 10$	$b = 10$	M2+S1
int $a = 10$;	$a = 0$	$b = 20$	M3+S1
int $b = 20$;	$a = 20$	$b = 20$	M4+S1
	$a = 40$	$b = 30$	M5+S1
$a = b$;	$a = 30$	$b = 30$	(M5+S3)/(M6+S3)/(M7+S3)/
$b = a$;			(M8+S3)
	$a = 30$	$b = 0$	M6+S1
	$a = 30$	$b = 50$	M7+S1
	$a = 0$	$b = 30$	M8+S1
	$a = 10$	$b = 20$	(M9+S1)/(M11+S1)/
			(M11+S3)
	$a = 20$	$b = 20$	(M10+S1)/(M2+S2)/(M4+S2)
	$a = 10$	$b = 10$	
	$a = 0$	$b = 10$	(M1+S2)/M3+S2)
	$a = 20$	$b = 0$	
	$a = 30$	$b = 20$	(M5+S2)/(M7+S2)
	$a = 10$	$b = 30$	
	$a = 0$	$b = 30$	(M6+S2)/(M8+S2)
	$a = 30$	$b = 0$	
	$a = 10$	$b = 20$	(M11+S2)
	$a = 10$	$b = 20$	

Fig. 2. Sample answer sheet

Participant code	Age	Sex	Time to do test	Prior programming	A-Level/s	Prior programming courses	Course result

Questions	Assignment								No effect	Equal sign	Swap values	Remarks (including participants' working notes)
	Assign-to-left		Assign-to-right		Add-Assign-to-left		Add-Assign-to-right		Values don't change (M9) / S	Assign means equal (M10) / S	Swap values (M11) /Ss / I	
	Lose-value (M1) /Ss / I	Keep-value (M2) /Ss / I	Lose-value (M3) /Ss / I	Keep-value (M4) /Ss / I	Keep-value (M5) /Ss / I	Lose-value (M6) /Ss / I	Keep-value (M7) /Ss / I	Lose-value (M8) /Ss / I				
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
C0												
C1												
C2												
C3												

Additional notes:

Fig. 3. The marksheet

Banff and Buchan college (UK) and at OSZ TIEM Berlin (Germany). The data for the first six of these experiments has been provided to us and is analysed here.

We divide the population according to whether they reported prior programming experience, and alternatively according to reported prior programming course attendance. In addition, based on reports of programming languages used, we classify programming experience as relevant to the test or not (essentially, whether or not subjects had been exposed to mechanisms of assignment and sequence similar to those used in the course they were about to take). Table 4 shows the effects of these background factors on success, shown as the percentage who answered yes and succeeded / the percentage who answered no and succeeded (the small number who did not reply in each case are ignored). There were no strong differences in the figures in any experiment: note, however, the extremely high success rates in the York experiment, and the slightly lower rates in Sheffield. The Westminster success rates are also fairly high. Some of these results were more significant than others: we don't comment on that at this point, but rely on meta-analysis.

Table 4. Effect of programming background on success in separate experiments

	NewC	Mdx1	Mdx2	Shef	West	York
Prior experience	63%/68%	55%/70%	51%/44%	93%/75%	75%/61%	92%/71%
Relevant experience	61%/66%	84%/55%	62%/42%	85%/78%	71%/69%	95%/88%
Prior course	63%/69%	69%/62%	44%/51%	72%/81%	74%/70%	90%/88%

To begin to look at the effect of consistency measured in the test on success in the course examination, we looked at the success rates of consistent subjects against the rest. As part of the improved experimental protocol, we were able to recognise levels of consistency: subjects who use a single model throughout are consistent; those who use two related models are also consistent but less so. We identified consistency levels C0, C1, C2 and C3, but in practice C0 was large and the others very small. Nevertheless we analyse the effect of consistency in two ways: C0/notC0 and C0-C3/notC. Table 5 gives the results.

Table 5. Effect of consistency on success in separate experiments

	NewC	Mdx1	Mdx2	Shef	West	York
C0/notC0	80%/44%	77%/54%	79%/27%	91%/47%	77%/60%	92%/50%
C0-C3/notC	79%/32%	70%/53%	64%/23%	91%/38%	75%/57%	90%/50%

In each experiment we found the same thing: consistent subjects did better than the rest. At York, as at Århus, most subjects scored consistently in the test (99 out of 105 in York, 124 out of 142 in Århus). There were so few non-consistent subjects at York that we could put little weight on that particular result, but we can, as we should, include it in a meta-analysis. In other experiments, apart from Mdx1 and Westminster, we find that consistent subjects do about twice as well as the rest. This discrepancy, and perhaps the York result as well, may be the effect of lenient examination. At Middlesex, where we have access to the examination materials, we know that Mdx1 was a weak first in-course quiz whereas Mdx2 was a stronger more technical second quiz: the second quiz separated students more radically and showed a stronger effect of consistency. The Westminster results were weaker even than Mdx1 and we wonder whether the examination was perhaps correspondingly lenient. At York the pass rate was 90%: lenient for that particular course population.

Table 6. Effect of consistency on success in subgroups, separate experiments

	NewC	Mdx1	Mdx2	Shef	West	York
CM2/notCM2	87%/56%	100%/58%	89%/41%	73%/81%	75%/70%	92%/78%
C0 (CM2 excluded)/notC0	74%/44%	67%/54%	74%/27%	97%/47%	77%/60%	92%/50%
Prior experience	79%/23%	75%/48%	88%/18%	93%/_	82%/62%	92%/100%
No prior experience	85%/33%	78%/62%	66%/33%	90%/47%	62%/60%	91%/0%
Relevant experience	79%/11%	93%/67%	100%/25%	86%/_	80%/50%	95%/100%
No relevant experience	80%/53%	63%/51%	70%/26%	92%/47%	84%/63%	88%/40%
Prior course	73%/33%	75%/55%	77%/31%	80%/0%	77%/70%	94%/50%
No prior course	93%/47%	81%/58%	86%/20%	94%/50%	93%/50%	88%/_

Table 7. Overall effect of programming background on success

	success yes/no	χ^2	*df*	p	**Significance**
Prior experience	74%/64%	21.02392	12	$0.05 < p < 0.10$	weak
Relevant experience	78%/68%	18.2602	12	$0.10 < p < 0.20$	very weak
Prior course	69%/73%	4.67478	12	$0.95 < p < 0.98$	none

Because the mark sheet identified not only consistency but also the particular model(s) used, we can recognise those subjects who apparently have already learned, before the course begins, the mechanisms of assignment and sequence that are used in the forthcoming course. In Java these models are M2 for assignment and S1 for sequence, and we called the group that consistently used these models in the test the CM2 group. Most, but not all of them reported prior programming experience. It would seem that most of the York intake could already program: 80 out of 105 were CM2. To see whether the effect of consistency was simply the effect of prior learning of programming, we looked at the population divided by CM2/notCM2, and we looked again at consistent subjects outside the CM2 group (C0 without CM2) against the rest. Then we looked at each of the subgroups defined by the background questions analysed in table 4. This gives eight population divisions, shown in table 6.

In *almost every cell* of this table the consistent group does better than the rest. Even in the lenient examination of Mdx1, at Westminster where a high proportion of non-consistent subjects passed, and at York where there were hardly any non-consistent subjects to consider, we see the same trend. The only cell which doesn't fit the picture is Sheffield CM2/notCM2 where the CM2 group (which was unusually small in that experiment) did worse than the rest, though it still did very well.

It is hard to grasp the overall message of this analysis by looking at individual results, which is why we turn to meta-analysis.

6 Meta-analysis

The Winer [34] procedure of meta-analysis was used to examine the overall effect of consistency and/or programming background on success. Winer's procedure combines p values from χ^2 analysis of separate experiments – the probability of obtaining the effect by accident – to give an overall p value. Because this is a meta-analysis of several experiments, our threshold significance value is set at a conservative 0.01 (1%).

Table 7 summarises the overall effects of programming background on success, showing the size of the effect, the χ^2 value and significance. None of the programming background factors had a large or a significant effect. The weak effect of prior programming experience and prior relevant experience was driven by 60% of candidates with prior programming experience overall, especially from the York experiment, an extreme case with 83% programming-skilful population.

Table 8. Overall effect of consistency on success

	success	χ^2	*df*	*p*	**Significance**
C0/notC0	84%/48%	49.84	12	$p < 0.001$	very high
C/notC	80%/42%	44.51	10	$p < 0.001$	very high

Table 9. Overall effect of consistency on success in filtered subgroups

	success	χ^2	*df*	*p*	**Significance**
CM2/notCM2	89%/62%	34.7474	12	$p < 0.001$	very high
C0 (CM2 excluded)/notC0	80%/48%	40.27092	12	$p < 0.001$	very high
Prior experience	84%/44%	31.64786	10	$p < 0.001$	very high
No prior experience	80%/47%	35.45406	12	$p < 0.001$	very high
Relevant experience	90%/35%	31.59638	10	$p < 0.001$	very high
No relevant experience	80%/50%	35.82052	12	$p < 0.001$	very high
Prior course	84%/50%	35.97697	12	$p < 0.001$	very high
No prior course	90%/46%	38.3514	10	$p < 0.001$	very high

Attending a programming course was shown to have no significant effect on success, both overall and in each individual experiment.

On the other hand, meta-analysis shows in table 8 a large and highly significant effect of consistency on success in both the C0/notC0 and C/notC group arrangements. Despite the weak effect in the first quiz at Middlesex and the Westminster experiment, especially in C/notC, none of the experiments is driving the result: if we eliminate any one of them there is still strong significance.

Table 9 summarises the overall effect of consistency on success in eight population divisions characterised by programming background factors. The overall result confirms the result of the initial experiment by demonstrating a highly significant effect of consistency on success in *every* slice. None of the experiments is driving the overall result: if we eliminate any one we still find significance.

This analysis shows that consistency is not simply the effect of learning to program. The CM2 group does do better than any other, as might be expected. But there are almost as many individuals who are C0-consistent but not CM2, their success rate is almost as good, and they are almost twice as likely to pass as those who are not consistent. Consistency is not simply the effect of learning to program.

The size of the effect varies according to the population division but it is significant everywhere and it is never small. Programming background, or its absence, doesn't eliminate the effect of consistency.

7 Conclusion and future work

The test characterises two populations in introductory programming courses which perform significantly differently. About half of novices spontaneously build and consistently apply a mental model of program execution; the other half are either unable to build a model or to apply one consistently. The first group perform very much better in their end-of-course examination than the second: an overall 84% success rate in the first group, 48% in the second (in C0/notC0 group arrangement). Despite the tendency of institutions to rely on students' prior programming background as a positive predictor factor for success, programming background has only a weak and insignificant effect on novices' success at best.

This study shows that students in the consistent subgroup have the ability to build a mental model, a drive to construct a system, something that follows rules like a mechanical construct,

and this is what more or less what Baron-Cohen's systematizers do. We speculate that we might be measuring a similar trait by different instruments, and Wray's results tend to support this.

Simon [27] stated "We do not pretend that there is a linear relationship between programming aptitude and mark in a first programming course, or that different first programming courses are assessed comparably; but we have succumbed to the need for an easily measured quantity." Although the test introduced by this study measures the ability to learn programming with some accuracy, without a solid method of measuring programming skill an optimum result cannot be achieved. Now we have a test which begins to measure programming aptitude, we need a standardised mechanism to measure programming achievement.

8 Acknowledgements

We would like to thank Mr. Ed Currie, Dr. Simon, Dr. Peter Rockett, Mr. Christopher Thorpe and Dr. Dimitar Kazakov, the collaborators who provided data for this study and our peers in PPIG (Psychology of Programming Interest Group) for their valuable advice.

References

1. S. Baron-Cohen, J. Richler, D. Bisarya, N. Gurunathan, and S. Wheelwright. The systemising quotient (SQ): An investigation of adults with Asperger's syndrome or high functioning autism and normal sex differences. *Philosophical Transactions of the Royal Society, Series B, Special issue on "Autism : Mind and Brain"*, 358:361–374, 2003. 3
2. S. Baron-Cohen, S. Wheelwright, R. Skinner, J. Martin, and Clubley. The autism spectrum quotient (AQ): Evidence from Asperger's syndrome/high functioning autism, males and females, scientists and mathematicians. *Journal of Autism and Developmental Disorders*, 31:5–17, 2001. 3
3. Myers M. VanBrackle L Beise, C. and L. Chevli-Saroq. An examination of age, race, and sex as predictors of success in the first programming course. *Journal of Informatics Education and Research*, 5(1):51–64, 2003. 2
4. Richard Bornat, Saeed Dehnadi, and Simon. Mental models, consistency and programming aptitude. In *ACE '08: Proceedings of the tenth conference on Australasian Computing Education*, pages 53–61, Darlinghurst, Australia, 2008. Australian Computer Society, Inc. 1
5. Michael E. Caspersen, Kasper Dalgaard Larsen, and Jens Bennedsen. Mental models and programming aptitude. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, pages 206–210, New York, NY, USA, 2007. ACM. 1, 3
6. E. Cross. The behavioral styles of computer programmers. In *Proc 8th Annual SIGCPR Conference*, pages 69–91, Maryland, WA, USA, 1970. 1
7. Saeed Dehnadi. Testing programming aptitude. In *Proceedings of the PPIG 18th Annual Workshop*, England, 2006. 1, 2
8. Saeed Dehnadi and Richard Bornat. The camel has two humps. In *Little PPIG*, Coventry, UK, 2006. 3, 4
9. Benedict du Boulay. Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1):57–73, 1986. 2
10. Jennifer L. Dyck and Richard E. Mayer. BASIC versus natural language: is there one underlying comprehension process? In *CHI '85: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 221–223, New York, NY, USA, 1985. ACM. 2
11. Sally Fincher, Raymond Lister, Tony Clear, Anthony Robins, Josh Tenenberg, and Marian Petre. Multi-institutional, multi-national studies in CSEd research: some design considerations and trade-offs. In *ICER '05: Proceedings of the first international workshop on Computing education research*, pages 111–121, New York, NY, USA, 2005. ACM. 2
12. Vikki Fix, Susan Wiedenbeck, and Jean Scholtz. Mental representations of programs by novices and experts. In *CHI '93: Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*, pages 74–79, New York, NY, USA, 1993. ACM. 2
13. C. M Kessler and J. R Anderson. Learning flow of control: Recursive and iterative procedures. *Human-Computer Interaction*, 2:135–166, 1986. 2
14. Raymond Lister, Elizabeth S. Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, Otto Seppälä, Beth Simon, and Lynda Thomas. A multi-national study of reading and tracing skills in novice programmers. In *ITiCSE-WGR '04: Working group reports from ITiCSE on Innovation and technology in computer science education*, pages 119–150, New York, NY, USA, 2004. ACM. 2
15. D.B. Mayer and A.W. Stalnaker. Selection and evaluation of computer personnel: the research history of SIG/CPR. In *Proc 1968 23rd ACM National Conference*, pages 657–670, Las Vegas, NV, USA, 1968. 1

16. Richard Mayer. *Thinking, Problem Solving, Cognition.* W. H. Freeman and Company Second Edition, New York, 2nd edition, 1992. 2
17. Richard E. Mayer. The psychology of how novices learn computer programming. *ACM Comput. Surv.*, 13(1):121–141, 1981. 2
18. L. P. McCoy and J. K Burton. The relationship of computer programming and mathematics in secondary students. *Computers in the Schools*, 4(3/4):159–166, 1988. 2
19. D. N. Perkins and R Simmons. Patterns for misunderstanding: An integrative model for science, math, and programming. *Review of Educational Research*, 58(3):303–326, 1988. 2
20. D.N Perkins, C Hancock, R Hobbs, F Martin, and R Simmons. Conditions of learning in novice programmers. *Journal of Educational Computing Research*, pages 261–279, 1989. 2
21. Ralph T Putnam, D Sleeman, Juliet A Baxter, and Laiani K Kuspa. A summary of misconceptions of high school BASIC programmers. *Journal of Educational Computing Research*, 2(4), 1986. 2
22. M. Raadt, M. Hamilton, R. Lister, J. Tutty, B. Baker, I. Box, Q. Cutts, S. Fincher, J. Hamer, P. Haden, M. Petre, A. Robins, Simon, K. Sutton, and D. Tolhurst. Approaches to learning in computer programming students and their effect on success. *Higher Education in a changing world: Research and Development in Higher Education*, 28:407–414, 2005. 2
23. Nathan Rountree, Janet Rountree, Anthony Robins, and Robert Hannah. Interacting factors that predict success and failure in a CS1 course. *SIGCSE Bull.*, 36(4):101–104, 2004. 2
24. R Samurcay. The concept of variable in programming: Its meaning and use in problem-solving by novice programmers. *Education Studies in Mathematics*, 16(2):143–161, 1985. 2
25. B. Shneiderman. *Software Psychology: Human Factors in computer and Information systems.* Winthrop, Cambridge, 1980. 2
26. Simon, Quintin Cutts, Sally Fincher, Patricia Haden, Anthony Robins, Ken Sutton, Bob Baker, Ilona Box, Michael de Raadt, John Hamer, Margaret Hamilton, Raymond Lister, Marian Petre, Denise Tolhurst, and Jodi Tutty. The ability to articulate strategy as a predictor of programming skill. In *ACE '06: Proceedings of the 8th Australasian conference on Computing education*, pages 181–188, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc. 2
27. Simon, Sally Fincher, Anthony Robins, Bob Baker, Ilona Box, Quintin Cutts, Michael de Raadt, Patricia Haden, John Hamer, Margaret Hamilton, Raymond Lister, Marian Petre, Ken Sutton, Denise Tolhurst, and Jodi Tutty. Predictors of success in a first programming course. *Eighth Australasian Computing Education Conference, Hobart*, 2006. 2, 9
28. E. Soloway and C. Spohrer, James. Lawrence Erlbaum Associates. 2
29. Maarten W. Van Someren. What's wrong? understanding beginners' problems with Prolog. *Instructional Science*, 19(4/5):257–282, 1990. 2
30. C. Spohrer, James and Elliot Soloway. Novice mistakes: are the folk wisdoms correct? *Commun. ACM*, 29(7):624–632, 1986. 2
31. Denise Tolhurst, Bob Baker, John Hamer, Ilona Box, Raymond Lister, Quintin Cutts, Marian Petre, Michael de Raadt, Anthony Robins, Sally Fincher, Simon, Patricia Haden, Ken Sutton, Margaret Hamilton, and Jodi Tutty. Do map drawing styles of novice programmers predict success in programming?: a multi-national, multi-institutional study. In *ACE '06: Proceedings of the 8th Australian conference on Computing education*, pages 213–222, Darlinghurst, Australia, 2006. Australian Computer Society, Inc. 2
32. P.C. Wason and P.N. Johnson-Laird. *Thinking and Reasoning.* Harmondsworth: Penguin, 1968. 2
33. Brenda Cantwell Wilson and Sharon Shrock. Contributing to success in an introductory computer science course: a study of twelve factors. *SIGCSE Bull.*, 33(1):184–188, 2001. 2
34. B.J. Winer, Donald R. Brown, and Kenneth M. Michels. *Statistical principles in experimental design.* Mc.Graw-Hill, Series in Psychology, 1971. 7
35. J.M. Wolfe. Perspectives on testing for programming aptitude. In *Proc 1971 26th ACM National Conference*, pages 268–277, Chicago, IL, USA, 1971. 1
36. Stuart Wray. SQ minus EQ can predict programming aptitude. In *Proceedings of the PPIG 19th Annual Workshop*, Finland, 2007. 1, 3